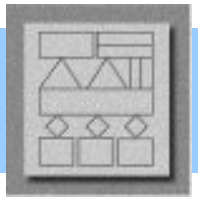


Data Management & Warehousing

WHITE PAPER

How Data Works

DAVID M WALKER

Version: 1.0
Date: 18/06/2007

Data Management & Warehousing

138 Finchampstead Road, Wokingham, Berkshire, RG41 2NU, United Kingdom

http://www.datamgmt.com

Table of Contents

Synopsis

Every business believes that their data is unique. However the storage and management of that data uses similar methods and technologies across all organisations. As a result the same issues of consistency, performance and quality occur across all organisations. The commercial difference between organisations is not whether they have data issues but how they react to them in order to improve the data.

This paper examines how data is structured and then examines characteristics such as the data model depth, the data volumes and the data complexity. Using these characteristics it is possible to look at the effects on the development of reporting structures, the types of data models used in data warehouses, the design and build of interfaces (especially ETL for data warehouses), data quality and query performance. Once the effects are understood it is possible for programmes and projects to reduce (but never remove) the impact of these characteristics resulting in cost savings for the business.

This paper also introduces concepts created by Data Management & Warehousing including:

- Left to right entity diagrams
- Data Model Depth
- Natural Star Schemas
- The Data Volume and Complexity graph
- Incremental Phase Benefit Model

Intended Audience

Reader	Recommended Reading
Executive	Synopsis
Business Users	Synopsis
IT Management	Synopsis and Introduction
IT Strategy	Synopsis
IT Project Management	Entire Document
IT Developers	Entire Document

About Data Management & Warehousing

Data Management & Warehousing is a specialist consultancy in data warehousing, based in Wokingham, Berkshire in the United Kingdom. Founded in 1995 by David M Walker, our consultants have worked for major corporations around the world including the US, Europe, Africa and the Middle East. Our clients are invariably large organisations with a pressing need for business intelligence. We have worked in many industry sectors but have specialists in Telco's, manufacturing, retail, financial and transport as well as technical expertise in many of the leading technologies.

For further information visit our website at: http://www.datamgmt.com

Introduction

Database systems hold the lifeblood of the organisation; the data with which decisions are made that affect every aspect of the business life. Whilst there are many technologies and methods in use the underlying characteristics for the storage and manipulation of data are very similar.

We intuitively understand that we have more transactions than we have customers because we assume that we will deal with at least some of our customers more than once. We know that the organisation structure will be affected by arrivals and departures of staff as well as by the growth or shrinkage of the company and that understanding who has responsibility for what over time will be important. We know that accurate geographic information is important whether it be for sales regions, delivery costs or market segmentation yet tracking customers' movements across geographies is difficult.

So why is it that we have such problems in managing the data for the organisation? What stops us from having high data quality and good performance on all our systems? The problems are mostly human in origin, for example:

- As customers we do not contact every company we have ever dealt with and provide them with our new address or other changes personal details.
- It takes time to communicate a new organisational structure.
- New systems are introduced to meet new business objectives, often with little thought to the issue of integrating data.

It is therefore impossible to design out the complexity of data; instead we must be proactive in dealing with the consequences of human interaction with systems. To do this we must understand how the data is modelled and stored and its effect on systems performance, data quality, interfaces (especially extraction transformation and loading for data warehouses) and data models for reporting systems.

This white paper describes the basics of how data is stored and looks at the consequences of that storage. It then examines how this knowledge can be used to improve the design and development of systems in general and specifically the benefits to data warehouse development.

How data is structured

Over the last 20 years relational databases[1] have become the norm for most commercial applications and data warehouses[2]. Even where the underlying storage of the database is non-relational[3] there is a need to provide the developer or user of the system with a relational interface. This allows users to write SQL[4] to create, retrieve, update and delete data from the system. It is effective because by design a well defined database will store one consistent version of each record. A well defined database is often referred to being in 'Third Normal Form.'[5]

Given this one might ask 'What are the reasons why such a large gap exists between operational database solutions and effective reporting solutions?':

- **Fit-for-purpose design**

  This is where the data that is stored and consequently the design of the database meets the business requirement of the application and nothing else.

- **Multiple Vendors/Developers**

  Since few organisations will acquire all their applications from a single source the approach taken to data modelling for the application will vary with each designer, developer, vendor, etc.

- **Table/Column Re-use**

  As systems grow over time the original purpose of a table or, more commonly, a column can be redefined. This redefinition can be a conscious decision or the result of a misunderstanding.

- **Human Error**

  Even if all other issues were resolved the design of a data model relies on individuals. Mistakes or inexperience can therefore lead to errors in the data model or more commonly in its interpretation.

These possible disparities in turn have two effects, one on the reporting of information and the other of the sharing of data between systems.

Reporting of data becomes reliant on the data stored by the developer. For example: an application may have a simple customer table that has the name of the customer and the customer status (either active or inactive). If the application was required to hold the status of the customer and the reporting was based around questions such as "What is the number of active customers?" then the application is fit-for-purpose. However as soon as the question "How many customers were active last month but are now inactive?" is asked then the data is no longer available because an update to the status field was made when the customer changed status and the historical value was discarded.[6]

---

[1] Relational Database: http://en.wikipedia.org/wiki/Relational_database
[2] A data warehouse is a database geared towards the business intelligence requirements of an organisation. The data warehouse integrates data from the various operational systems and is typically loaded from these systems at regular intervals.
[3] For example Sybase IQ which uses column rather than row based storage.
[4] SQL, or Structured Query Language: http://en.wikipedia.org/wiki/SQL
[5] A definition of 'Normal Forms' can be found in Appendix 2 – Database Normalisation.
[6] Obviously, as with all the simple examples used in this paper, there are easy solutions for any particular problem. The examples are given to show what can happen.

The second effect is in the sharing of information between databases. Here there are a number of problems that arise from differing data models. The first occurs with interfaces, the sharing of information between two systems. Each system might have a different data model and therefore the information has to be translated between them.

For example two systems store information about the customer's children. In one system the number of children is stored as a count e.g. 2, whilst in the other system two records are created, one for each child e.g. John and Mary. It is easy to pass the required data from the system that holds the names of the children to the system that holds the count. The data is created by counting the number of child records the customer has. However the converse is not true. Even though the system knows that a customer has a child it still cannot supply the required information which is the child's name.

In fact the designer may have chosen to store the data in a number of ways depending on the needs of the system:

- As a property: Customer has one child.
- As an event: Customer had a child on 1st January 2000.
- As a hierarchy: Customer has child 1, Customer has child 2.
- As a relationship: Customer is the parent of Child.

The sharing of data between systems is also affected by the timing of the arrival of the data. In our example above one system may be updated to say that a particular customer has had a second child, the other system still thinks that the customer only has a single child. When the interface is run which data value should take precedence? The answer is obvious in the case of the example but this may be running in an environment where there are thousands of updates happening concurrently on each of the two systems every day.

Sharing data between two systems is obviously complex enough, however large organisations often run data warehouses as their reporting systems and with them a specialist type of interface called Extract, Transform and Load (or ETL). The data warehouse will have a data model that normally has the following characteristics:

- The data model will be different from all the data models in the source systems by design.
- The data model will be populated with data copied from many different systems.
- The data model will store historical data which will necessarily be incomplete.
- The data model will support the handling of differences in the timing of the arrival of data.

The behaviour of data that this paper examines is true for both transactional and data warehousing systems that use relational models, snowflakes and star schemas, although how they are affected varies in degree.

The example database model

This paper now needs to make use a data model[7] in order to explain some concepts. For those not familiar with data models they are simply a diagrammatic representation of the information within a database.

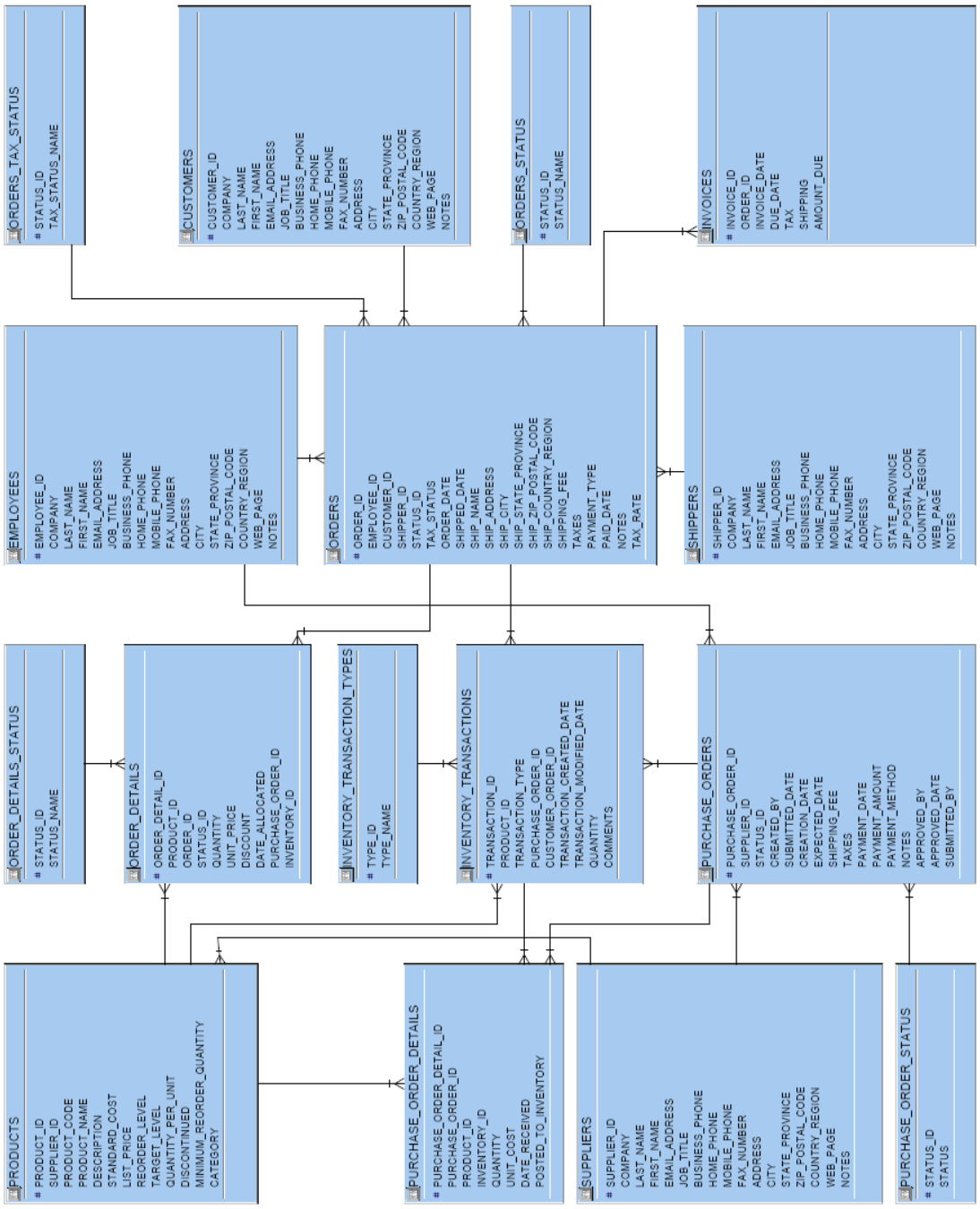


Figure 1 - Standard Entity Relationship Data Model

[7] The data model used is a slightly modified version of the OLTP Northwind demonstration database that is provided with free Microsoft Access. It is used not because it is an example of good or bad data modelling but because it is representative of a fairly standard relational data model for a transactional system.

This model shows a small business with customers, suppliers and products and then the purchase from suppliers and the orders and invoices raised for customers. The representation as traditionally drawn helps understand the content of the tables and their relationships but does not help explain how the data will work. The important characteristic of the data model is the relationships and their 'cardinality'. A relationship is often drawn using the one of the following representations:

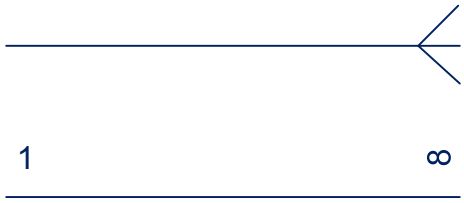


Figure 2 - One to Many Relationships

This simply means that the table connected to the 'one' will have a record related to 'many' records in the other table, e.g. one customer has many orders, one transaction type has many transactions, etc. The side that has the one relationship is referred to as having the primary key, whilst the side having the many will be referred to as having the foreign key. A table can only have one primary key, which has a unique set of values but it can have many foreign keys, where values are duplicated.

Given these standard definitions we can now consider redrawing the standard entity relationship diagram into a 'left to right' entity relationship diagram[8]. This is done by following a simple set of rules.

1. Put all tables that have primary keys and no foreign keys on the left hand side of the diagram in a column.

   In our example this includes Suppliers, Purchase Order Status, Invoice Transaction Types, Customers, Order Status, Order Tax Status etc.

2. Move all the tables in the first column that are not reference data across into a second column. Those left behind are in level one, whilst the others are in level two.

   In our example level two includes Customers, Suppliers, Shippers, Employees.

3. Move all the remaining tables into the third and subsequent levels such that the table with the primary key is always in a level to the left of the table with a foreign key. The table must be placed in the left most level that satisfies this requirement for all its foreign keys.

Doing this for the entire data model will produce a left to right entity relationship diagram such as the one on the next page.[9]

Experienced data modellers should see Appendix 3 – Resolving Specific Left To Right Issues to help with this process.

[8] A term and method developed by Data Management & Warehousing for laying out a data model in such a way as to support analysis of the data structures.
[9] For clarity only columns used in either primary or foreign keys are shown, all others have been hidden.

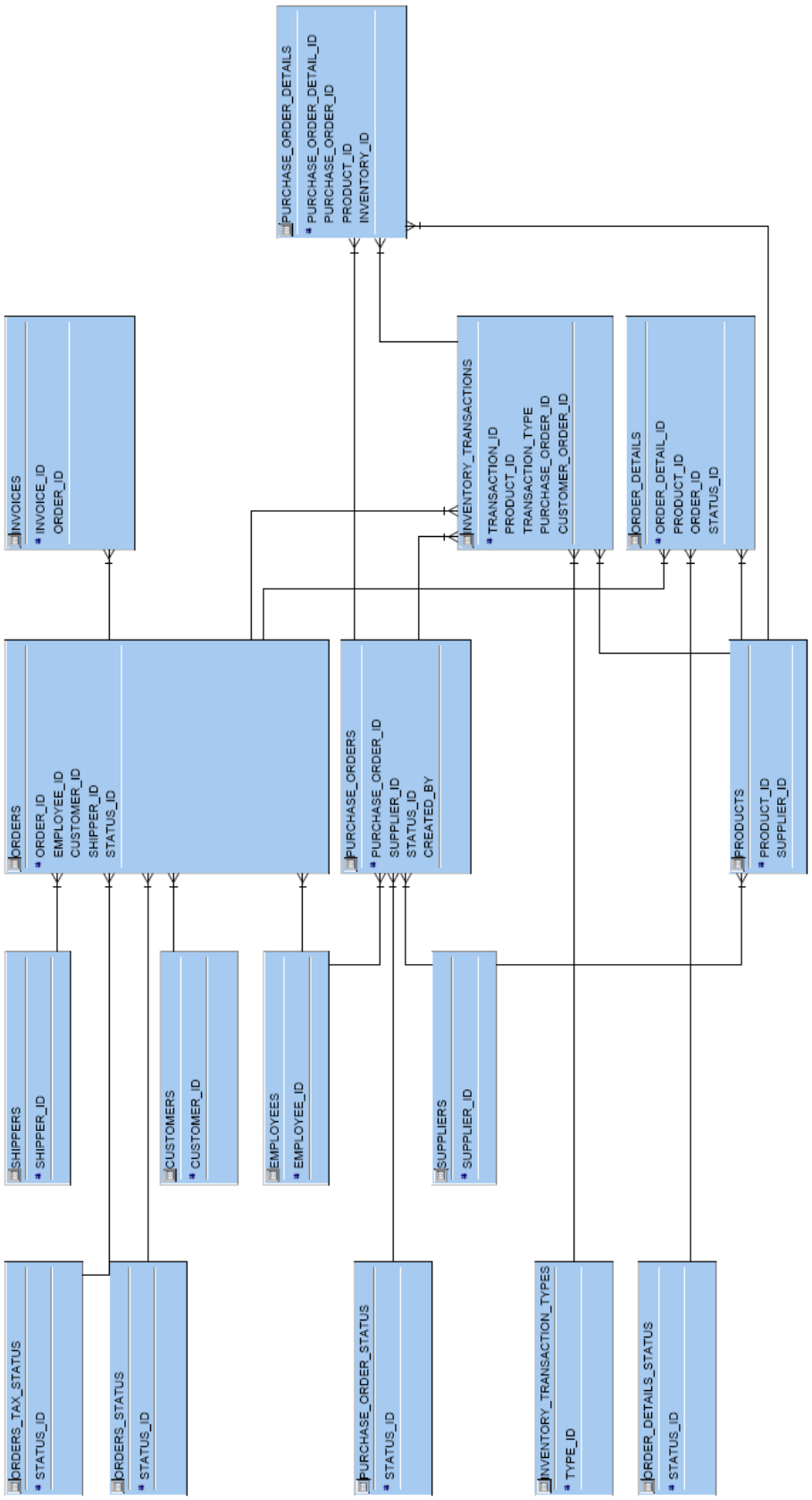


Figure 3 - The Left-to-Right Entity Relationship Diagram

Left to Right Entity Relationship Diagrams

Once the left to right entity relationship diagram is created there are some consequences that must be considered to understand how data works:

Data Model Depth

The data model depth is defined by the number of levels that the left to right entity relationship diagram has:

- A simple data mart star schema used in a data warehouse will, by definition only have two levels in which the dimensions exist in the first level and the facts in the second level.
- A snowflake schema with have a depth of three or four, simply because it takes the star schema and extends individual dimensions with outrigger tables.
- OLTP and Data Warehouse systems will have a depth of between five and seven even for the most complex of systems. This number rarely rises above eight.

It is often surprising to those that have not previously used this technique that regardless of industry, application, developer or underlying technology this basic structure of data remains true.

Furthermore each level will have progressively less tables in it. The first level will have the most, the second level normally less than half the first level, right down to the last level which may have only one or two tables in it. This is to be expected because the tables to the left are being used to define those to the right with 'one-to-many' relationships.

Volume and Complexity Graph

The data model depth allows us to consider the volume of data that the system will have. By definition each table going from left to right has more information than its predecessor. The first level will have many small tables, typically tens or hundreds of rows. The next level will contain less but larger tables whose size relates to the type of business that organisation is in.[10] Subsequent levels get larger but there are fewer tables.

There is also a direct relationship to the complexity of the data. Complexity of data is defined by its accuracy and completeness. Data such as customer names and addresses is both inherently incomplete and often contains data errors caused by the difficultly in capturing the data[11] accurately and reliably. Transactional data however is automatically generated and therefore contains significantly fewer data quality issues. For example a wrongly dialled telephone number still generates a valid call data record, a supermarket cashier who scans a product twice by accident corrects it by signifying a minus and rescanning the item creating a new negative transaction and a bank error is not edited but a counter or reverse entry is created and then the correct entry inserted. The left to right entity relationship diagram levels can be plotted in terms of volume and complexity.

[10] A list of industry examples can be found in Appendix 4 – Industry Typical Volumes
[11] This is known as the data acquisition process

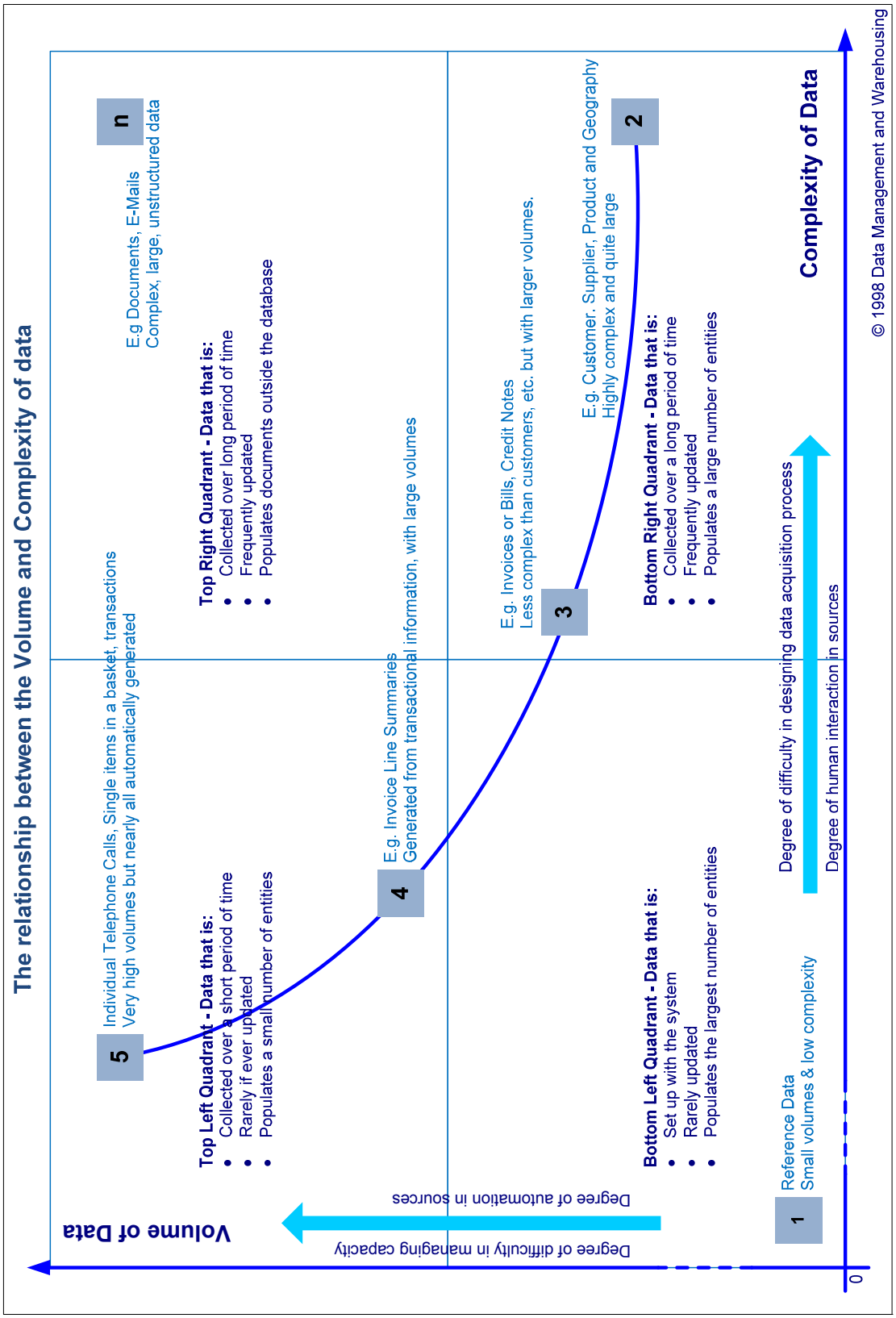


Figure 4 - Volume and Complexity of Data

The figure shows many characteristics of how data works:

- Managing the overall storage requirement of the system relates directly to the number of transactions held and how they are summarized, all other data will, in total, be smaller.

- The largest volume of data is the most automated and therefore less subject to updates and reaches a point where it is complete and can be archived.

- The most difficult data to get right from the data quality point of view will be the customers, products, geography, etc because it is the most complex. This extends from the design phase (Which columns are used to describe product? How many address lines are required?) through the build stage (How to manage addresses with do not conform to the data model?) to the usage stage (Users who just don't read that the postcode must be in field four, or products that are now characterised by colour instead of size).

- Complex data is often being updated and rarely if ever reaches a state of completeness, for example whilst one might archive transactions over three years old one would not want to remove the customers who made those transactions in case they came back. If all the volume is contained in the transaction data then a consideration is 'What is the lost opportunity cost when compared to the storage cost of archiving the customer data set?'

- Tables in level one have few columns. Tables in level two have few foreign keys and many columns ('short and fat' tables). Each subsequent level normally has fewer columns and more keys than the previous level ('long and skinny' tables).

- Even reference data is not static but changes with time, e.g. in 2006 the UK added the concept of a civil marriage to the list of possible relationship values (single, married, divorced, widowed) and there is an insurance company that hold gender as MM, MF, FM, FF – the values are for Born Male, Now Male; Born Male, Now Female; Born Female Now Male; Born Female, Now Female because life expectancy relates to birth gender, whilst marketing required current gender.

- The figure also describes a level 'n' of high volume, high complexity data. This is the arena of unstructured data and the information explosion of the last few years. Whilst this area is a hot topic for many organisations[12] it is out of the scope of this paper and probably outside the capability of many organisations who have yet to get to grips with their structured data.

The left to right entity relationship diagram and the volume and complexity graph can now be used as part of the toolkit for project managers, technical architects and developers when building systems.

[12] There are many new regulations for organisations that require them to hold information such as emails and documents and be able to search and retrieve them in order to demonstrate compliance, as well as for internal audit purposes.

Master Data Management

Master Data Management (MDM)[13], also known as Reference Data Management, is the management of reference or master data that is shared by several disparate IT systems and groups. Master Data Management is required to enable consistent information between diverse system architectures and business functions.

Large companies often have IT systems that are used by diverse business functions (e.g., finance, sales, R&D, etc.) and span across multiple countries. These diverse systems usually need to share key data that is relevant to the parent company (e.g., products, customers, and suppliers). It is critical for the company to consistently use these shared data elements through various IT systems.

Master Data Management is, by definition, the management of data held in Levels 1 and 2 of the left to right entity diagram of an enterprise wide data model. This is a deceptively simple statement that raises a number of issues:

- Does the enterprise have a data model that describes the entire business? If not then how does the organisation define what master data needs management?
- Which systems hold master data? It is not necessary for a single system to hold the master data for all entities, for example customer may be held in one system, whilst product may be held in another system.
- Are all the attributes of a master data entity held in the same system? For example a customer name and address may be held in one system and the customer telephone number is held in another system, the combination of which holds the entire customer record.

We can describe the type of master data that we have as follows:

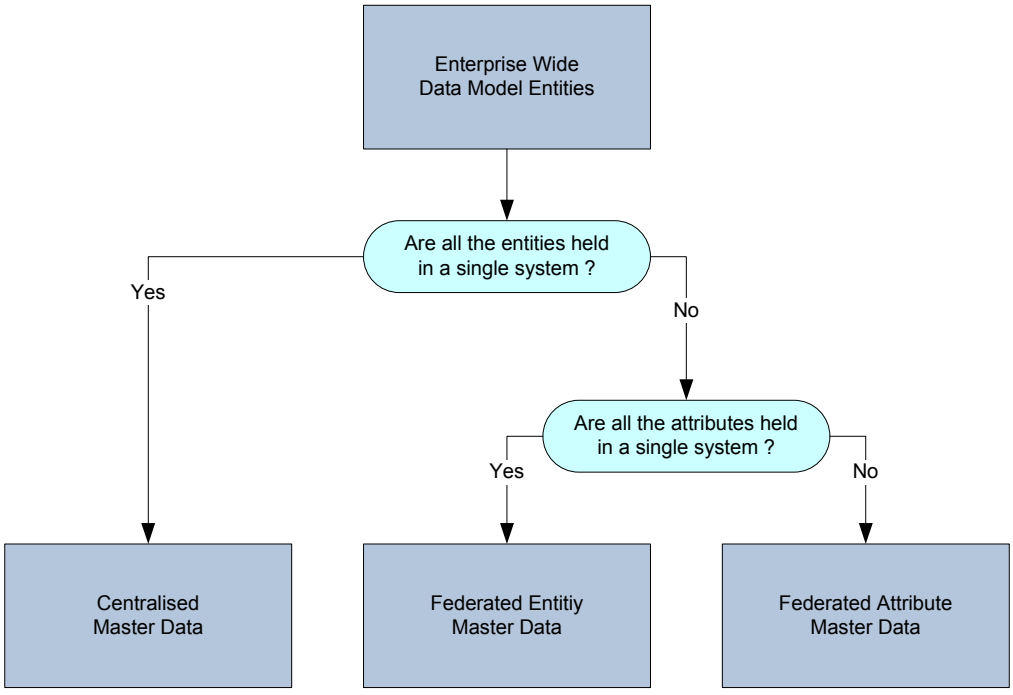


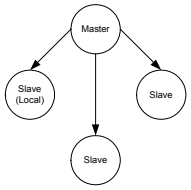
Figure 5 - Types of Master Data

[13] Definition from Wikipedia: http://en.wikipedia.org/wiki/Master_Data_Management

It is also possible to categorise how this master data is managed in terms of the creation and updating of data. This can take one of three forms:

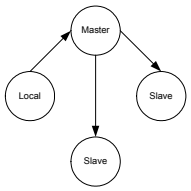
- **Master -> Slave updates**

  This is where the update can only occur in the master and then every other system is updated from the master.



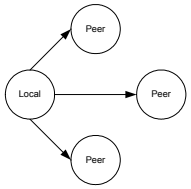
- **Local -> Master -> Slave updates**

  This is where the local system is updated first, the update is then passed to the master which subsequently updates all slaves.



- **Peer-to-peer updates**

  This is where the local system is updated and then it updates all other systems that hold copies of the data.



These updates are either done in real time using a messaging system or via a batch interface using ETL technologies.

There are advantages and disadvantages to all these of these approaches:

	Master -> Slave	Local -> Master -> Slave	Peer -to-Peer
Advantages	Single well defined master Single place for an update to occur	Single well defined master Local transactions have no latency for the user	No single point of failure Lower end-to-end latency between systems
Disadvantages	Requires all inserts and updates to be intercepted on local systems Introduces latency for the user whilst waiting for the master to be updated Single point of failure	Introduction of latency between the local and master being updated Contradictory updates can occur in local systems Single point of failure	Lacks a single well defined master. Source system responsible for multiple updates Contradictory updates can occur in local systems

Since an organisation's IT systems are built up incrementally over time it is likely that any current infrastructure will be made up of systems that aspire to use a number of different types of master data and a number of different update models. Since both data quality and accurate reporting depend heavily on good master data it is essential that organisations develop the appropriate processes and infrastructure for its management.

Implementing a Master Data Management strategy can therefore be broken down into a number of steps:

- Identify the master data entities and attributes that need to be managed.
- Identify the systems where the current master data is held.
- Assess data quality issues within the current data sets.
- Identify the type of master data solution that is most suitable for the organisation.
- Identify the types of master data updates that are appropriate for the organisation.
- Define policies and procedures for updating the data.
- Identify technology gaps in order to implement policies and procedures.
- Action initial data quality issues.
- Implement technologies.
- Implement policies and procedures.

It should be noted that the above can be broken down into a number of phases so that groups of entities can be implemented rather than approaching the problem as a single massive implementation.

This sort of programme of work often runs side by side with data warehousing projects which have a pre-requisite for developing strong master data management methods. The governance is also similar to that of a data warehouse project and can be run along similar lines.[14]

Using the steps above it can be seen that Master Data Management is about an architecture, policies and procedures designed to integrate and maintain information rather than deploying specific technologies. Whilst Master Data Management software, Messaging Hubs and ETL tools are critical in supporting the implementation of master data management they are not 'silver bullets' that will solve the issues relating to data consistency.

If an entity in the enterprise data model does not exist in any system then it is necessary to create a system to maintain the data. This will either be as part of the master data management system or as part of the warehouse support application (WSA).[15]

[14] Data Management & Warehousing publish a white paper on Data Warehouse Governance which is available from: http://www.datamgmt.com/index.php?module=article&view=78
[15] For further information about Warehouse Support Applications see Overview Architecture for Enterprise Data Warehouses which is available from: http://www.datamgmt.com/index.php?module=article&view=76

Using these techniques for Data Warehousing

The sections above can provide developers with a toolkit that can have a significant impact on the success of a data warehouse[16]. The following sections highlight just some of the ways in which this toolkit can be used:

Natural Star Schemas

Data Warehouse projects often have a series of data marts that have data structured in a star schema for reporting purposes. Much is written on how to design and build star schemas but using the left to right entity relationship diagram can be a significant aid. Left to right entity relationship diagrams highlight the "natural star schema"[17] within the model. An examination of the diagram shows that some tables on the right hand side have only foreign keys, these are the "natural facts". In the example tables such as Purchase Orders, Invoices and Order Details are all natural facts.

Tracing back from the natural fact through all the relationships will describe all the information required for the dimensions. In the example Order Details is related to Products, Order Details Status and Orders. The Orders table is related to Employees, Customers, Shippers, Orders Status and Orders Tax Status. The Products table is related to Suppliers. These related tables will become the basis of the dimensions.

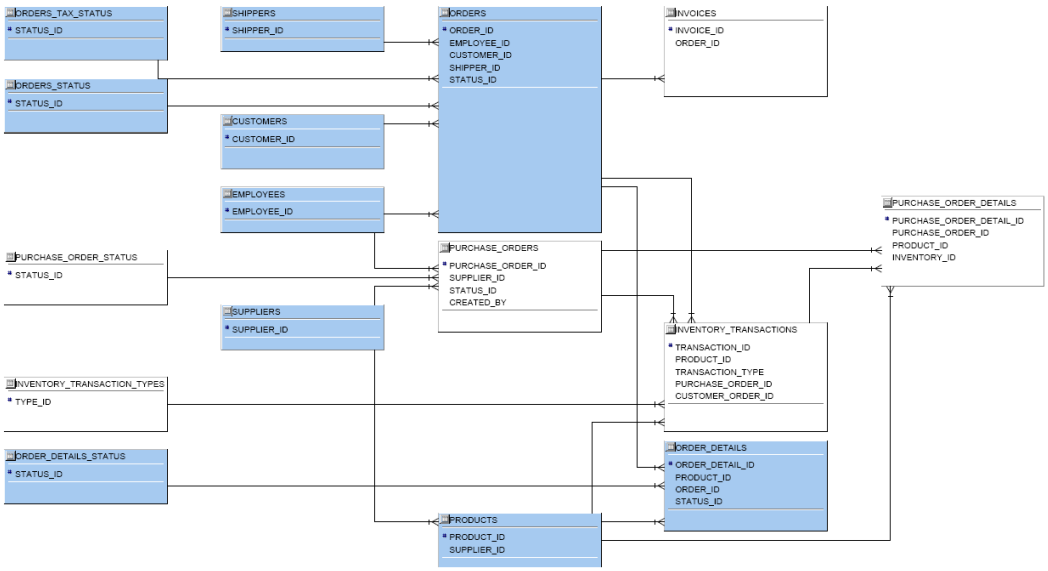


Figure 6 - The tables that make up the Natural Star Schema

In the example closer scrutiny shows that the fact table would benefit from being a join of the Orders and Order Details to form a single fact table and the supplier key should be 'de-normalised' into the new fact table and the shipping information would be normalised out into a separate dimension. The remaining tables would form eight further dimensions.

[16] Data warehousing projects often use many overlapping sets of terminology. The terminology used here is consistent with those defined in our other white papers including "Overview Architecture for Enterprise Data Warehouses."

[17] Natural Star Schemas are a term devised by Data Management & Warehousing to explain star schemas that occur in data models as a result of data structures rather than explicit data mart design activities.

Other small changes are needed in the dimensions to create a production data mart. Tables in level one are normally Type 1 dimensions[18]. Tables in other levels normally become Type 2 dimensions requiring start date and end date fields to be added. Finally, in the example the designer has chosen to omit certain columns that are not appropriate for reporting (e.g. fax number).

This process provides a fairly functional first cut of the data mart derived directly from the data model when presented as a left to right entity relationship diagram. The example source data model does not have enough tables to show the full impact of this technique but the benefits increase with larger schemas.

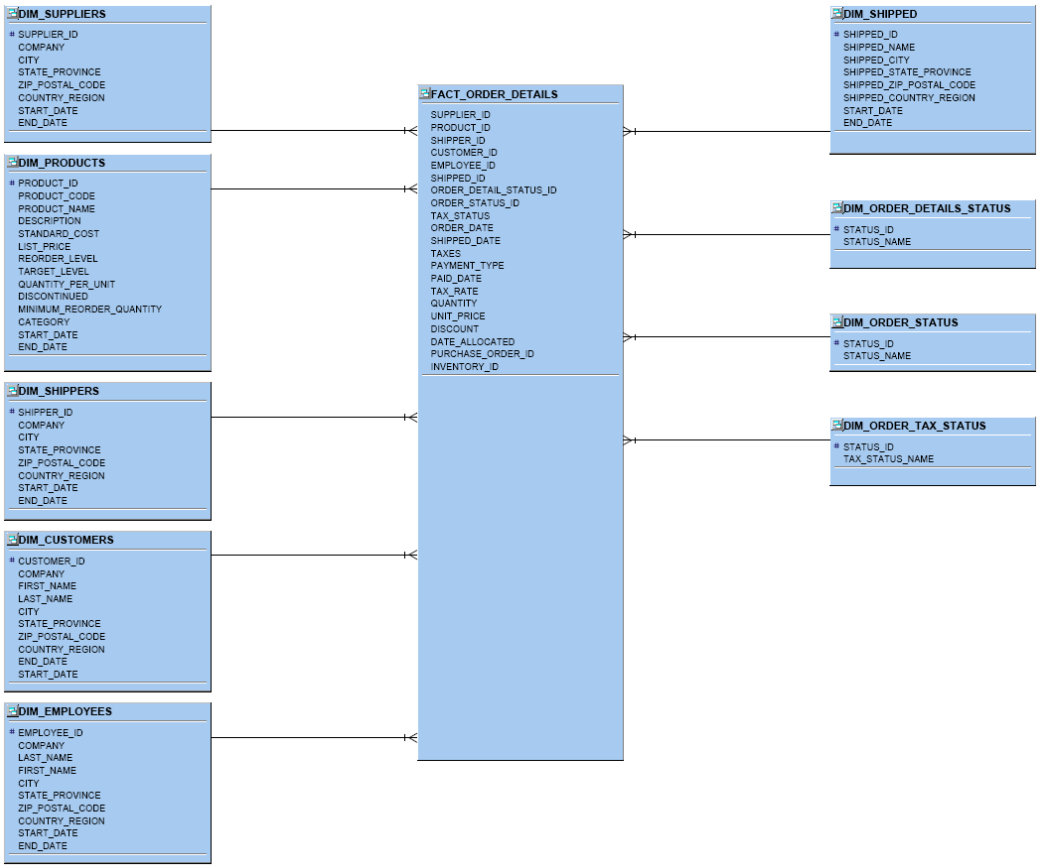


Figure 7 - Star Schema

[18] A description of data mart dimension types can be found at: http://en.wikipedia.org/wiki/Dimension_table and was first described by Kimball, Ralph et al (1998); The Data Warehouse Lifecycle Toolkit, p17. published by Wiley.

Data Models for the Data Warehouse

The previous section has shown that it is relatively easy to find natural star schemas in a data model. This often raises questions about the architectural approach and type of data model design which should be used. Using the above techniques the following guidelines can be given:

- **Departmental Data Marts**

  Characteristics: A small number of star schemas fed from a single source.

  Technique: Design star schemas derived from the natural star schemas in the source system model and load via a staging area.

- **Enterprise Data Warehouse with Dependent Data Marts**[19]

  Characteristics: An enterprise data model fed from multiple source systems with data marts built from the enterprise data model.

  Technique: Design the enterprise data model following normal data modelling procedures and then design the data marts from the natural star schemas in the enterprise data models.

- **Data Warehouse Appliance**

  Characteristics: A relational data model either based on an enterprise model or on one of the source systems and augmented with data from other source systems on a bespoke data warehousing platform.

  Technique: A single relational model in the database with natural star schemas implemented in the reporting tool meta-layer.[20]

Extract, Transform & Load (ETL)

The left to right entity relationship diagram also has a number of effects on how the ETL is built and how it is put together in a schedule to run.

The first and most simple aspect to note is that tables used in level one must be loaded before those in level two and so on. This is because the referential integrity requires that foreign keys must pre-exist. This can be avoided by turning off referential integrity in the database but exposes the system to data quality problems and is to be avoided.

The second feature relates to the effort required to populate the table. Complex tables and large tables will take more time than simple reference tables. This means that early on in the development of the ETL for a project that there will be a significant impact on development times because of the complex data that needs to be handled.[21]

---

[19] This is described in some detail in:
The Overview Architecture for Enterprise Data Warehouses
http://www.datamgmt.com/index.php?module=article&view=76

[20] Meta-layers are parts of the reporting tool that isolate underlying database from the end users. e.g. the 'Universe' in Business Objects and the 'End User Layer' in Oracle Discoverer, etc.

[21] A worked example of the impact on ETL development effort of the Volume and Complexity graph can be found in Appendix 5 – ETL Effort Example.

The third consideration is in the skills and tools needed to build the ETL. Complex data requires more updates and more sophisticated data cleansing e.g. for cleaning names and addresses, whilst large volumes of data will have few if any updates, minimal data cleansing but require a tool capable of very fast load times. Business analysts will be able to help with the rules for complex data and DBAs will be needed to support the developers for large tables that need partitioning, and performance tuning.

The fourth item to consider is the scope of any given phase of the ETL. Each phase of the project should ideally have the objective of delivering a data mart.[22] By identifying the natural star schema that will support the data mart it is possible to size the number of tables in a particular phase and the number of phases required. Also some tables (especially level one and level two tables) will be used by multiple phases.

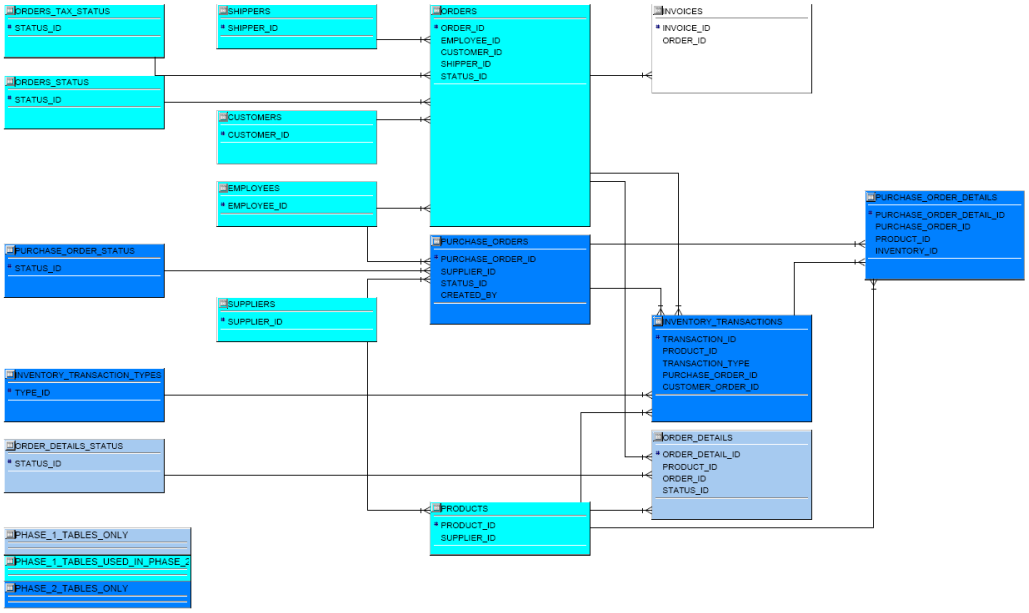


Figure 8 - Phase 2 Incremental Benefits

Even in our simple example the initial phase would have used ten tables, the second phase would only use five new tables and re-use data from eight of the phase one tables.

This means that the first phase to populate a table pays the standard cost, but subsequent phases that reuse tables have an incremental benefit not only in shorter development times because there are fewer tables to populate but also in completeness of range of tables available to query.

This is described by Data Management & Warehousing as the 'Incremental Benefits Pyramid.

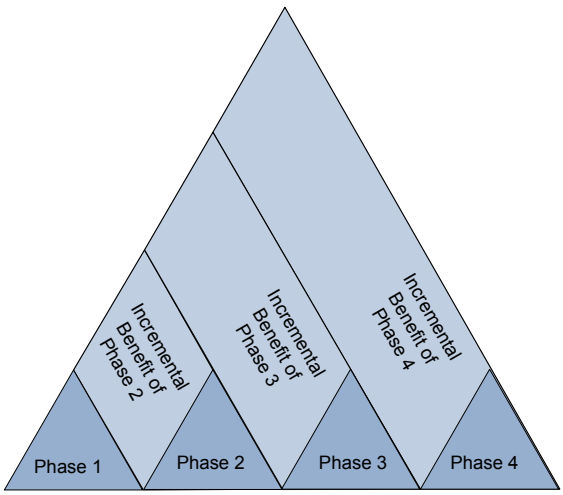


Figure 9 - Incremental Benefits Pyramid

[22] Some projects prefer to deliver multiple data marts per phase. Both the Agile Methodology and Data Management & Warehousing recommend having more, smaller phases rather than fewer larger phases as this helps maintain project momentum.

Data Quality

The left to right entity relationship diagram has been used to break down tables into different levels. The type of data quality issue that affects a given table is often related to the level.

Data Quality and human interaction

The level two and level three tables are those that are most affected by user input. These are tables which contain information such as individual and organisations names, addresses, and product descriptions. This data often relies on individuals to key the information.

Keying errors result in inconsistencies and data that has subtle differences. For example 'Data Management & Warehousing' and 'Data Management and Warehousing' would not match and result in undercounting.

Many of these errors can be corrected by rule based data cleansing[23], however some information needs more sophisticated techniques, especially for addresses.[24]

Data Quality and the management of hierarchies

Many of the tables in levels two and three are associated with hierarchies, e.g. product hierarchy, organisational (employee) hierarchy, customer hierarchy, etc. These hierarchies are often ragged (or unbalanced)[25] and frequently changing.

For example, the hierarchy of the organisational structure is modified from time to time and has a changing association with the people that fulfil these roles. These changes are often held in presentation tools on a shared drive rather than as data in a source system. The hierarchy may be used to determine commission or for the aggregation of other KPIs and is further complicated by the comings and goings of staff within an organisation.[26]

Hierarchies are used to aggregate data for reporting. Small errors in tracking the changes to the hierarchy lead to disproportionately large errors in summary reports.

[23] An example of rule based cleansing can be found in Appendix 6 – Rule Based Cleansing.
[24] In the UK the Postcode Address File, or PAF, is the most up-to-date and complete address database in the UK, containing over 27 million addresses. http://www.royalmail.com/portal/rm
[25] A ragged hierarchy is one in which the number of levels and the number of leaves within a level are not identical for all branches of the tree.
[26] In 2006 the average UK staff turnover rate was 18.3%. For a 750 person organisation this rate would represent a person joining/leaving the organisation every other working day. (http://www.cipd.co.uk/subjects/hrpract/turnover/empturnretent.htm).

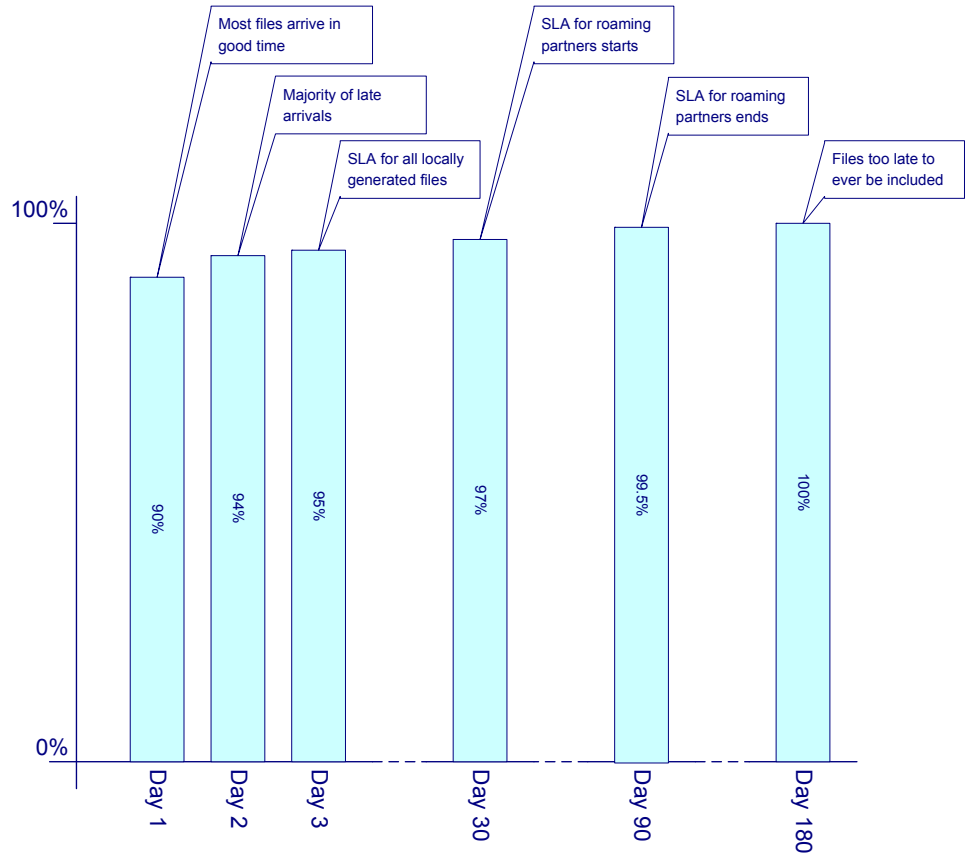
Data Quality and data latency

Data latency is a measure of the time taken for data to reach the system after it is created. It can be broken down into two sub types: ad-hoc data latency and systemic data latency.

Ad-hoc data latency is related to the time taken for people to inform systems of changes, e.g. changes of address, availability of new products, etc. This data is found in levels two and three. It will always have completeness issues and normally can only be improved by incentivising those who create/provide the data at source.

Systemic data latency is a result of the processes required to load the data. This affects tables in level five and beyond of the left to right entity relationship diagram. For example a supermarket chain that requires each of its stores to report the daily transactions, or a telephone company that requires all the call data records (or CDRs) from its own switches and those of its roaming partners.

Whilst the data is generated immediately there may be system components between the source and the target that fail or do not have sufficient bandwidth to deliver the information in time. In this case the data is normally loaded on subsequent days, however for a period of time all the 'available' data will have been loaded but the data will still be incomplete.



Example of Call Data Record (CDR) delivery in a Telco
(Based on our consultants experience between 1995 and 2007)

Figure 10 – Example Telco Systemic Data Latency

Data Quality and de-normalised data

De-normalisations (where data is copied to a table breaking the rules of normalisation) are often included in data models to increase performance. De-normalisation is a valid technique in the design of databases but has data quality consequences. The process opens up the system to inconsistencies as multiple versions of the de-normalised data can occur.

By definition de-normalisation can not occur in level one, but is common in levels two, three and four. De-normalised data should be normalised and cleansed as part of the data warehouse load process.

Data Quality and disabled constraints

The building of a left to right entity relationship diagram relies on a series of primary to foreign key constraints. These constraints can either be enforced or disabled in the source system. They are commonly disabled in the higher level tables in order to improve performance.

A system that has disabled constraints is open to any data being added to the foreign key without validation against a primary key. In order to fix this issue surrogate records have to be generated in the table that holds the primary key. This may in turn cause duplicate information in the table holding the primary key that has to be resolved in order to get a consistent set of data.

Data Quality and algorithmic error

Tables in level four will often suffer from algorithmic error. This is where the source system carries out a process such as bill generation. The data warehouse has both the transaction files and the bills as generated, i.e. it is a repository of generated information and not a creator of the information itself. There is a temptation to try and reconcile the bills in level four with the transactions, this is nearly always impossible.

The bill is generated with a specific (and often complex) algorithm using reference data at a particular point in time. If the reconciliation process does not use exactly the same algorithm and the same data then the results will differ. The cost of exactly reproducing and maintaining the algorithm from one system in another will always be prohibitive.

Data Quality and systems migration

If a reporting system is a replacement for an existing reporting system then one of the most common problems is the reconciliation of data between the two before allowing the new system to ‘go live’.

If the new system has followed the design requirements and been daisy-chain tested[27] but differs from the original system which one is right? It is desirable to chase down the discrepancies but it might never be possible to eliminate all of them, and as the original system is being replaced one has to assume that the new system is correct, and the original system has some flaw that has previously gone un-detected.

[27] See Appendix 7 – Daisy Chain Testing for a description of this approach

The sections above show that different types of data quality issues affect different levels within the data warehouse and need to be addressed within the context of where they appear in the data model.

Performance

The ideas behind the left to right entity relationship diagram also have an effect on the performance of a system. Whilst database platform specific considerations are outside the scope of this particular paper some general observations can be made:

Data Storage

There are currently two major categories of data storage within databases used for data warehousing. The traditional and by far the most common is row based,[28] where data is stored as a single record. The second type is column (or vector) based where data is held in its columns and then the row is made up of a record of pointers to the data in the columns.[29]

Row Based Storage

Row based storage is often relatively expensive in terms of disk space, normally requiring a multiplier of the raw data to store it effectively. More disk means greater I/O time required and slower response. The higher the level of the table in the left to right entity diagram the more likely it is that the table will need size management techniques, e.g.:

- **Indexes**
  Identify records in a set quickly at the cost of additional disk space
- **Compression**
  Store records in compressed format but more CPU is required to de-compress the data on the fly at query time.
- **Partitioning**
  Split the data into multiple partitions to enable parallel query at runtime but this also requires more CPU and slightly more disk space.
- **Aggregates**
  These use more disk space and CPU time in advance of the query in order to gain runtime performance on specific queries.

Row based technologies however are very good at managing concurrent updates because each record is held in its entirety in one location.

[28] Database such as Oracle, Microsoft SQL Server, Sybase ASE, Netezza
[29] Sybase IQ is the largest proponent of this method, but other vendors also exist.

Column Based Storage

Column based storage is less expensive in terms of disk space as any repeated values are stored in a very small space (e.g. take a field (American) 'State' in the database with 50 million addresses. In a row based database this would take at least 425Mb but only 50Kb in a column structure. This gives a large performance gain in terms of I/O and disk storage required.

However there is a significant impact in the cost of updates by comparison with row based databases. Firstly the individual update is difficult to manage from a database perspective as both the column and the vector have to be updated and secondly this normally locks the entire table there can be significant concurrency issues.

When using column based technologies for updates it is therefore worth considering generating the updated data sets outside the database and in the ETL tool and performing the minimum update work inside the database. This can lead to a completely different ETL design.

The tables with the most records on the right hand side of the left to right entity relationship diagram deliver the biggest benefit from using column based storage as they are low on updates and have lots of low cardinality data[30] in a low number of columns.

Indexing

Databases now provide a whole range of indexing options[31] but deploying indexes should be done with care.

The first indexes that will be used on the database are those used to enforce referential integrity. These are normally B-tree indexes. Beyond these indexes there is little benefit in creating any more indexes for tables on the left or right of the diagram. Those in levels two, three and four will benefit from bitmap and hash indexes after analysis of the types of queries that are being made on them.

Indexes should be used sparingly[32] as they consume additional disk space and slow the insert/update process, however they do significantly improve the user experience when querying.

As can be seen there are a number of techniques that in specific circumstances can help performance however they are all a trade offs and good data warehouse design is about striking the right balance rather than absolutes when choosing techniques.

[30] Low cardinality data is data in which the range of possible values is small e.g. States in the United States of America, by comparison with high cardinality data such as forenames.
[31] Some databases such as Netezza do not use indexes but use massive partitioning as an alternative strategy.
[32] Historically data warehouses have often used an 'index everything' strategy. This is now less common as un-indexed query performance has improved and the cost of building and maintaining indexes in terms of CPU and disk space used has risen in line with the growth in data volumes.

Summary

An understanding of how data is modelled and stored provides a valuable insight into how to manage the issues that arise. This white paper has looked at the basic structure of information, the nature of the one-to-many relationship and the consequences of that in terms of volume and complexity.

From this it has been possible to develop the simple technique of left to right entity relationship diagrams and use this to identify the characteristics that affect data quality, performance and the use of certain types of data models, especially in the data warehousing environment.

Businesses today are dealing with the problems of data explosion. A typical business today (2007) stores ten times more data than in 2000 and Gartner estimates that storage requirements will have increased by a factor of thirty by 2012. The concept of 'one size fits all' management of information will not scale to meet the demand.

Some of the problems in handling all this new information will be dealt with by new algorithms for querying and better methods of storing the data. It will not deal with the underlying issues of data quality and relative performance for specific business queries. The ability to break down the problems with data into discreet categorisations and develop specific techniques to deal with these problems is the first step towards a solution.

An organisation that has this understanding and can exploit it in the development of systems will have a significant competitive advantage because it will derive more value from the data available to it, whilst also being able to afford to maintain the data.

Appendices

Appendix 1 – Entities and Tables

Throughout this document there have been references to both entities and attributes as well as tables and columns. In general entities and attributes are descriptive of logical data modelling whilst tables and columns are used to describe physical modelling.

A logical entity is normally described by a physical table in the database, whilst each attribute is described by a column.

Whilst the techniques described can be applied to the logical model they are normally applied after the creation of the physical models and hence this document has referred to tables and columns rather than entities and attributes.

Data modelling tools tend to refer to all diagrams as entity relationship diagrams regardless of whether they are logical or physical and again this document has stuck to that convention.

Appendix 2 – Database Normalisation

The normal forms[33] of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is to such inconsistencies and anomalies. Each table has a “highest normal form": by definition, a table always meets the requirements of its highest normal form and of all normal forms lower than its highest normal form; also by definition, a table fails to meet the requirements of any normal form higher than its highest normal form. The normal forms are applicable to individual tables; to say that an entire database is in normal form n is to say that all of its tables are in normal form n.

Edgar F. Codd[34] originally defined the first three normal forms (1NF, 2NF, and 3NF). These normal forms have been summarized as requiring that all non-key columns be dependent on "the key, the whole key and nothing but the key.” The fourth and fifth normal forms (4NF and 5NF) deal specifically with the representation of many-to-many and one-to-many relationships among columns.

First normal form

The criteria for first normal form[35] (1NF) are:

- A table must be guaranteed not to have any duplicate records; therefore it must have at least one candidate key.
- There must be no repeating groups, i.e. no columns which occur a different number of times on different records.

[33] This appendix is an edited form of: http://en.wikipedia.org/wiki/Database_normalization
[34] Date, C. J. (1999), An Introduction to Database Systems (8th ed.). Addison-Wesley Longman. ISBN 0-321-19784-4. http://www.aw-bc.com/catalog/academic/product/0,1144,0321197844,00.html
[35] Kent, W. (1983) A Simple Guide to Five Normal Forms in Relational Database Theory, Communications of the ACM, vol. 26, pp. 120-125; http://www.bkent.net/Doc/simple5.htm.

Second normal form

The criteria for second normal form (2NF) are:

- The table must be in 1NF.
- None of the non-prime columns of the table are functionally dependent on a part (proper subset) of a candidate key; in other words, all functional dependencies of non-prime columns on candidate keys are full functional dependencies.

Third normal form

The criteria for third normal form (3NF) are:

- The table must be in 2NF.
- There are no non-trivial functional dependencies between non-prime columns. A violation of 3NF would mean that at least one non-prime column is only indirectly dependent (transitively dependent) on a candidate key, by virtue of being functionally dependent on another non-prime column.

Fourth normal form

The criteria for fourth normal form (4NF) are:

- The table must be in 3NF.
- There must be no non-trivial multi-valued dependencies on something other than a super-key. A 3NF table is said to be in 4NF if and only if all of its multi-valued dependencies are functional dependencies.

Fifth normal form

The criteria for fifth normal form (5NF) are:

- The table must be in 4NF.
- There must be no non-trivial join dependencies that do not follow from the key constraints. A 4NF table is said to be in the 5NF if and only if every join dependency in it is implied by the candidate keys.

Appendix 3 – Resolving Specific Left To Right Issues

Experienced data modellers will have a number of questions on specific relationship types that are not covered in the general description above.

1. Q. What about one-to-one relationships?

   A. These are put into the same level provided all the other foreign key relationship rules allow such a placement.

2. Q. What about many-to-many relationships?

   A. This can be dealt with by creating a surrogate 'resolving' table that contains all foreign key from each of the two tables and therefore exists in the level to the right of the two tables.

3. Q. Our database doesn't enforce referential integrity, what should we do?

   A. This process is about the model rather than the database, so create the relationships in the model even if they are not enforced in the database.

4. Q. What about tables that contain summary information?

   A. If these tables contain only summary information that can be found elsewhere then they can be ignored otherwise they should be included.

5. Q. What about de-normalisations?

   A. De-normalisations should be included but review the section in the main document on data quality.

6. Q. What about security model tables e.g. the privileges table in the example?

   A. These can be excluded but for completeness are normally left in.

Appendix 4 – Industry Typical Volumes

The table below outlines some typical industry[36] values to show how the volume of data is related to the data model depth. The factor is the multiplying factor that each level has on the previous level's data volume. Note that the volume in first level is unrelated to the final volume of data.

Telco

Level	*Table*	*Factor*	*Database Rows*
1	Reference Data		1,000
2	Subscribers	1	20,000,000
3	Bills per year	12	240,000,000
4	Summary lines per bill	3	720,000,000
5	Calls per summary line	20	14,400,000,000

Retail Supermarket

Level	*Table*	*Factor*	*Database Rows*
1	Reference Data		1,000
2	Customers	1	30,000,000
3	Baskets per Year	52	1,560,000,000
4	Summary lines per Basket	2	3,120,000,000
5	Items per basket	15	46,800,000,000

Bank

Level	*Table*	*Factor*	*Database Rows*
1	Reference Data		1,000
2	Customers	1	10,000,000
3	Statements per year	12	120,000,000
4	Summary Lines per Statement	3	360,000,000
5	Transactions per Statement	20	7,200,000,000

Wholesale Outlet

Level	*Table*	*Factor*	*Database Rows*
1	Reference Data		1,000
2	Customers	1	100,000
3	Invoices per Year	24	2,400,000
4	Product Categories per Invoice	2	4,800,000
5	Items per Invoice	10	48,000,000

Airline

Level	*Table*	*Factor*	*Database Rows*
1	Reference Data		1,000
2	Customers	1	1,000,000
3	Flights per Year	6	6,000,000
4	Legs per flight	2.1	12,600,000
5	Changes per flight	1.1	13,860,000

[36] Averaged information from UK service providers based on a national population of around 60 million people taken in 2006.

Appendix 5 – ETL Effort Example

Projects often get the effort for developing ETL wrong causing significant project overruns. This is often because they make no allowance for the volume and complexity relationships or the dependencies that affect the order in which things are built.

A typical model might have one hundred tables to populate and assign the effort to build each level incrementally (e.g. 2 units of work for level 1, 4 for level 2, etc.). This method of assignment is wrong and will lead to large overruns.

Level	Tables	Units of Work	Effort	Cum. Effort
1	50	2	100	100
2	25	4	100	200
3	12	6	72	272
4	8	8	64	336
5	5	10	50	386

Figure 11 - Typical estimate (usually an under-estimate)

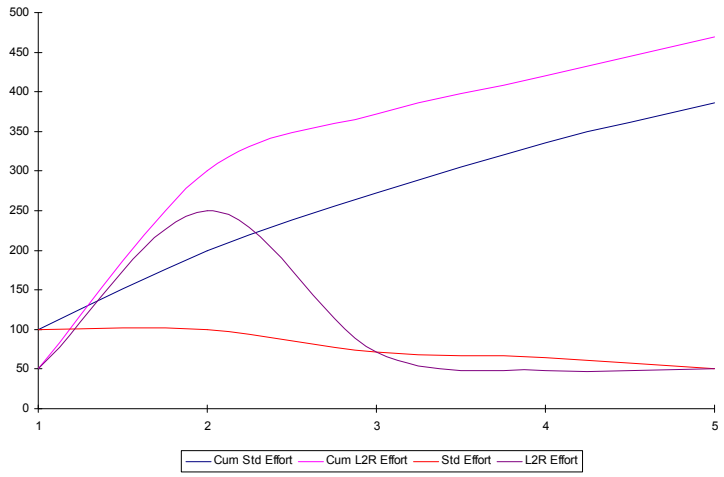
If, however, values from the volume and complexity graph are used and assign higher units of work to complex or large volume data then the effort turns out significantly different.

Level	Tables	Units of Work	Effort	Cum. Effort
1	50	1	50	50
2	25	10	250	300
3	12	6	72	372
4	8	6	48	420
5	5	10	50	470

Figure 12 – More accurate estimate using volume and complexity

The second method predicts that the total effort will be 20% higher than the first method and predicts significant time spent dealing with complex data.

If the reader has experience of a data warehouse project that has overrun then they may have heard a statement like:



'We started well and got ahead of where we thought we would be, then we hit some real data quality and performance issues that caused a long delay before we finally got back on track. However we were never able to claw back the time we lost when dealing with the problems.'

Compare the statement with the values in the cumulative effort of each estimating technique and it becomes clear that it is the complex tables that have the impact on the delivery timescales.

Appendix 6 – Rule Based Cleansing

Rule based cleansing is relatively simple to write as part of the ETL development.[37] The following simple example shows the effect of a small number of rules on a set of data.

Firstly it is recommended that any rule based cleanser has two columns, one for the source data and one for the clean version.

Company Name	Clean Company Name
Data Management & Warehousing	
Data Management and Warehousing	
Data Management and Warehousing Ltd	
Data Mgmt and Warehousing	

Rule 1: Copy the data to the clean column

Company Name	Clean Company Name
Data Management & Warehousing	Data Management & Warehousing
Data Management and Warehousing	Data Management and Warehousing
Data Management and Warehousing Ltd	Data Management and Warehousing Ltd
Data Mgmt and Warehousing	Data Mgmt and Warehousing

Rule 2: Make all the data upper case
(As this field is used for comparison all cases should be the same)

Company Name	Clean Company Name
Data Management & Warehousing	DATA MANAGEMENT & WAREHOUSING
Data Management and Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing Ltd	DATA MANAGEMENT AND WAREHOUSING LTD
Data Mgmt and Warehousing	DATA MGMT AND WAREHOUSING

Rule 3: Replace ‘ & ‘ with ‘ AND ‘
(This should be done with all symbols e.g. replace ‘%’ with 'per cent' and develop rules appropriate for other punctuation such as commas and full stops)

Company Name	Clean Company Name
Data Management & Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing Ltd	DATA MANAGEMENT AND WAREHOUSING LTD
Data Mgmt and Warehousing	DATA MGMT AND WAREHOUSING

Rule 4: Remove ‘ LTD’ from all records
(Standard abbreviations should be removed or replaced with the long version e.g. removing PLC from company names replacing RD with ROAD, etc. Note that it is useful to replace either Saint or Street with ST rather than try and determine the long name that should be used.)

Company Name	Clean Company Name
Data Management & Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing Ltd	DATA MANAGEMENT AND WAREHOUSING
Data Mgmt and Warehousing	DATA MGMT AND WAREHOUSING

Rule 5: Replace all double spaces with a single space
(Other useful punctuation management includes trimming leading and training white space and removing tabs etc.)

Company Name	Clean Company Name
Data Management & Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing	DATA MANAGEMENT AND WAREHOUSING
Data Management and Warehousing Ltd	DATA MANAGEMENT AND WAREHOUSING
Data Mgmt and Warehousing	DATA MGMT AND WAREHOUSING

The result is imperfect but significantly better than the original data.

[37] Data Management & Warehousing have a complete rule based engine processor for various platforms including all solutions using Oracle databases. The tool works by holding the rules in a metadata table and then applying them when called from the ETL tool.

Appendix 7 – Daisy Chain Testing

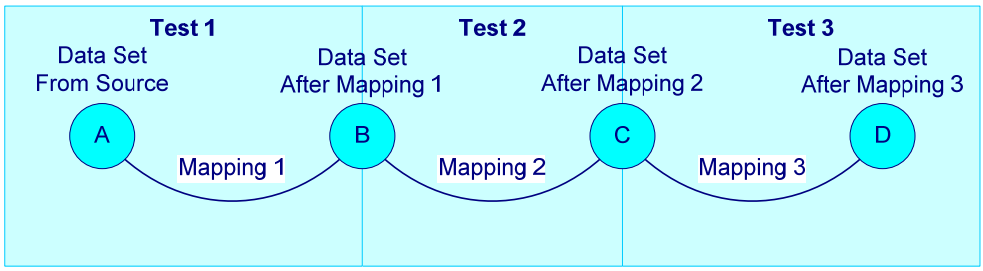
The process of testing the data warehouse is often seen as an insurmountable task. This is because so much of the data that goes into the system is manipulated, integrated, aggregated and transformed that a direct comparison with a source system is impossible.

The most successful approach to testing the system is known as daisy-chain testing. In this approach the functional correctness of each step is used to validate the whole.

For example loading a table requires three ETL mappings then there is there is a need to create a test for each step that looks at:

- The entry criteria - What data has to be available?
- The exit criteria – What data should be produced?
- The boundary conditions – What are the special cases?

A test is then carried out for each mapping:



There is no direct test that can compare A to D but if Test 1 (A to B), Test 2 (B to C) and Test 3 (C to D) are correct then A to D has to be correct.